웹 환경에서 유연성 있는 작업 할당을 위한 가상 병렬 처리 시스템 개발

정권호[†] · 송은하[††] · 정영식[†††]

요 약

웹은 네트워크로 연결된 모든 컴퓨터를 하나로 묶는 거대한 가상 시스템을 구성한다. 인터넷에 존재하는 수많은 유휴 상태 시스템을 이용하여 병렬 처리함으로써 비용 대 성능비가 매우 높으며 강력한 컴퓨팅 파워를 요구하는 거대한 문제를 해결할 수 있다. 하지만, 로컬 네트워크가 아닌 인터넷 전체를 대상으로 하는 글로벌 환경에서 병렬 수행하는데 호스트들간의 이질성, 접근의 용이성, 작업에 대한 신뢰성을 고려해야 한다. 본 논문은 가상 병렬 처리 시스템인 WebImg를 설계 및 구현하여 웹 컴퓨팅이 가능하며 동일한 작업을 여러 호스트에게 분배하기 위한 유연성 있는 작업 할당 전략을 제시하고 그 성능을 평가한다. 작업에 참여한 이기종 호스트들이 가변적인 환경에서 작업 수행 도중 시스템의 성능변화에 대처하여 재할당 연산을 이용한 유연성 있는 작업 할당 기법을 제시한다. 더욱이 제안한 작업 할당 전략은 참여 호스트의 상태를 수시로 제어하여 결함내성을 제공한다.

Development of Virtual Parallel Processing System for Flexible Task Allocation on the Web

Kwon-Ho Jung[†], Eun-Ha Song[††] and Young-Sik Jeong[†††]

ABSTRACT

Web consists of the grand virtual system which is made of all connected computers network. We can solve the huge problem which requires high quality in cost performance and powerful computing power to use a numerous idle state system on internet as process it parallel. However, we have to consider heterogeneous computing resources, accessibility, and reliability to carry out parallel system on global environment, not network but whole Internet. In this paper, We the WebImg system which has the power of web computing, and show the flexible task allocation strategy in heterogeneous hosts. Also, we evaluate its performance, moreover the proposed task allocation strategy supplies fault tolerance by controlling host situation at any time.

1. 서 론

1.1 연구 배경 및 목적

컴퓨터 네트워크 기술과 웹 기술의 발전으로 사용자들의 수가 기하급수적으로 증대되고 있다. 이와 비례하여 다양한 정보를 제공받기 위한 사용자의 요구사항 또한 질적 · 양적으로 높아져 응용 분야가 넓어졌으며 요구되는 계산량도 늘어나게 되어 거대한 컴퓨팅 파워가 필요하게 되었다. 하지만 값비싼 고성능 슈퍼컴퓨터보다 다수의 컴퓨팅 자원을 사용하여 많은 작업량을 지닌 애플리케이션을 나누어 수행하는 병렬 처리로 인해 빠른 시간에 작업을 마칠 수 있게 되었으며, 전체 작업의 수행시간을 단축시킬 수 있게 되었다[1-3].

인터넷의 성장과 함께 고성능화되어 비교적 단순히 기계적인 작업에 사용되고 있는 개인 컴퓨팅 자원

본 논문은 2000년 원광대학교 교비지원에 의해서 연구됨
† 원광대학교 컴퓨터공학과 대학원
†† 준회원, 원광대학교 컴퓨터공학과 대학원
††† 정회원, 원광대학교 컴퓨터 및 정보통신공학부 부교수

의 다수가 네트워크로 연결되면서 고단위의 지능 있는 작업을 할 수 있다는 연구가 오래 전부터 이루어져 왔다. 그러나, 이것은 LAN이라는 지역적으로 한정된 곳에 연결된 여러 호스트들을 이용한 연구들이 대부분이었다[4,5]. 하지만, 이제는 시·공간적인 제한점을 극복하여 전세계의 인터넷에 연동되어 있는 자원이라면 어떠한 정보로 얻을 수 있게 되었으며 지구상의 연결되어 있는 모든 자원을 자신이 처리하는 작업에 이용할 수 있는 환경으로 바뀌어가고 있다.

그러므로 웹은 무수히 많은 호스트들을 연결하는 가장 큰 가상 시스템(Virtual System)이 되고 있다. 가상 시스템은 거대한 하나의 컴퓨터로 볼 수 있으며 이러한 계산 환경을 글로벌 컴퓨팅(Global Computing)이라고 한다. 글로벌 컴퓨팅에 존재하는 무수한 컴퓨터들의 유휴시간(idle time)을 이용하여 대규모의 병렬 처리를 수행하여 비용대 성능비(cost performance)가 뛰어난 컴퓨팅 환경을 구성할 것으로 기대된다. 그러나, 이러한 환경은 이질적인(heterogeneous) 컴퓨팅 자원으로 구성되어 있으므로 컴퓨팅 자원들 각각의 플랫폼에 맞는 컴파일링 작업이 추가로 요구되며 제공자와 사용자간의 신뢰성(reliability)이 떨어진다. 뿐만 아니라 컴퓨팅에 참여한 호스트의 수, 다양한 호스트들의 성능 및 상태, 이들의 물리적인 거리등을 예측할 수 없으며 호스트 수에 비례하는 성능 증대와 작업 결과에 대한 응답시간을 예상할 수 없다. 또한 컴퓨팅 자원으로 이용하기 위한 이들의 제어 방법이 없다는 문제점을 갖고 있다.

따라서, 본 논문에서는 앞서 언급한 이러한 문제점을 해결하는 가상 병렬 처리 시스템을 제안한다. 자바 수행 환경(Java execution environment)은 플랫폼 독립성을 보장하여 글로벌 컴퓨팅에 참여하는 네트워크 상의 호스트들의 이질성을 극복하게 해주는 좋은 도구가 되고 있다. 특히 웹 인터페이스에 내장된 자바 가상 머신(JVM)[11]위에서 애플릿(applet) 형태로 수행되어진다.

널리 산재되어 있는 자원들을 빠른 네트워크로 연결하여 단말 사용자가 하나의 애플리케이션에 대하여 병렬 수행을 위해 몇 개의 작업으로 분할하고 이들 작업들을 서로 다른 호스트에게 할당하는데 있어서 그들의 성능에 맞는 작업을 할당하며, 특히 할당받은 작업을 수행 도중에 호스트의 가용성(system availability)에 따라 가변적인 성능 이상 상태 발생으로 연산 지연(latency), 더욱이 진행중의 고장(failure)으로 더 이상 연산할 수 없는 호스트의 부분 작업을 성공적으로 끝마칠 수 있도록 재할당 연산을 수행하는 유연성 있는 작업 할당(Flexible Task Allocation) 기법을 제시한다.

본 논문에서 제시하는 시스템은 작업 요청자(Job Requester : JR)/작업 관리자(Job Manager : JM)/작업 처리자(Job Processor : JP)로 구성한다. 작업 서비스 요청을 시작하면서 자바가 지원하는 RMI (Remote Method Invocation)을 이용하여 JR 참조값을 전달하기 때문에 JP는 작업 결과를 JM 중재없이 직접 전달한다. 그러므로 하나의 작업 관리자가 가질 수 있는 오버헤드가 발생하지 않으며 전체의 수행시간도 줄일 수 있다. 또한 시스템을 관리하는 작업 관리자는 작업 관찰자(Job Observer : JO)를 두어서 작업에 대한 결과 정보를 얻어 요소들간의 신뢰성을 얻는다. 구현된 가상 병렬 처리 시스템의 활용으로 프랙탈 이미지 처리(Fractal Image Processing)를 적용하여 제안한 기법의 성능을 평가한다.

1.2 관련 연구

본 논문과 관련되는 시스템으로 ATLAS[4]는 자바와 Clik의 프로그래밍 기술을 접합하였으며 네트워크 기반의 자원들을 활용하여 병렬 수행할 수 있도록 설계되었다. 작업 취득 스케줄링(Work-stealing Scheduling)알고리즘을 사용하여 균형적인 작업 할당 전략을 지원하고, 트리 구조의 프로그래밍 모델을 가진다. 그러나, 트리 구조에서 오는 적용 가능한 애플리케이션의 제한, 플랫폼 종속적인 라이브러리의 사용의 단점을 가지며 결함내성에 대한 언급을 하지 않았다.

Charlotte[5]은 순수 자바를 사용하여 인터넷에 연결된 어떤 컴퓨터라도 웹 브라우저만 있으면 자원을 제공할 수 있도록 설계된 자바 기반 병렬 플랫폼이다. 열정적 스케줄링(eager scheduling)과 2단계 멱등 수행 전략(two-phase idempotent execution strategy)을 사용하여 결함내성과 균형적인 작업 할당 전략을 제공한다. 하지만 작업들 사이의 통신을 지원하기 위한 가상 공유 메모리 방법으로 같은 작업을 가진 여러 호스트의 접근으로 통신량의 증가에 따른 과도한 자원 낭비가 발생할 수 있으며 매니저의 오버헤드 문제가 있다. ParaWeb[6]은 자바 병렬 클래스 라이브

러리(Java Parallel Class Library)의 구축을 통해 메시지 전달 방법을 사용하여 웹 컴퓨팅의 자원을 활용할 수 있는 해결책을 제시하였으나 균형있는 작업 할당을 위한 해결 방법은 제시하지 못했다. Javelin[9]은 낙관적 작업 전파(optimistic load propagation)에 기반한 간단한 작업 할당 기법을 제시하였으며 형식론에 기반한 성능분석은 제시하지 않았다. SuperWeb[7]은 브로커에 의해 호스트들의 등록 및 관리와 작업 스케줄링을 하므로 브로커의 오버헤드로 확장성이 부족하다. Popcorn[8]은 호스트의 고장으로 미수행된 computelet를 발견하자마자 단순한 방법을 통해 다른 호스트에게 대신 작업을 요구하는 방법으로 결함내성을 제공하며 요구받은 computelet의 실행을 빨리 마친 호스트에게 다른 computelet을 맡기는 방법으로 작업 할당 전략 사용하였다. 그러나 이 전략은 작업의 결함내성과 균형적인 작업 할당 전략은 미흡하며, 하나의 Market 시스템이 시작에서 결과까지 관리하므로 병목 현상이 발생되기 쉽다. JET[10]는 체크포인팅(checkpointing)과 로깅(logging) 기법을 적용해서 주기적으로 저장하는 방법을 사용하여 결함내성을 제공하였으나, 작업 할당 전략에 관한 알고리즘은 제시하지 않았다.

본 논문에서 개발하는 WebImg 시스템은 원격 객체 참조 기법을 이용한 구현으로 경제성을 고려한다. 통신에 대한 JM의 중재로 인한 오버헤드를 줄이는 메커니즘을 적용하고 자바 기술을 이용하여 WebImg 시스템의 호환성과 확장성을 유지하도록 하였다. 또한, 웹 환경이 미치는 가변적인 요인에 효과적으로 대처하기 위한 유연성 있는 작업 할당 기법을 통해 결함내성과 균형있는 작업 할당 전략을 제공한다.

2. WebImg 설계

2.1 WebImg 구조

웹 환경은 다양한 컴퓨터들로 연결되어 있는 하나의 가상 시스템이다. 가상 시스템의 참여자를 다음과 같이 구분하며 WebImg 시스템의 구성요소라고 정의한다. 특정한 애플리케이션을 수행하고자 많은 자원을 요구하는 작업 요청자(Job Requester : JR), 자신의 자원을 제공하여 연산을 수행하는 작업 처리자(Job Processor : JP), JR과 JP를 관리하는 작업 관리자(Job Manager : JM), JP의 작업 처리 상태를 관측하는 작업 관찰자(Job Observer : JO)이며 이들의 관계는 다음 그림 1과 같다.

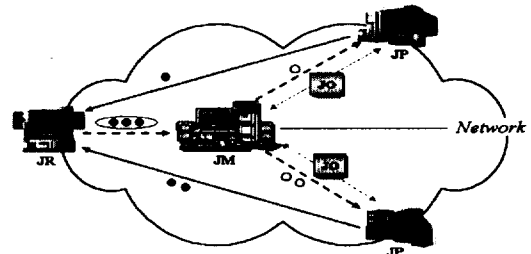



그림 1. WebImg의 개념적 구성도

2.2 WebImg 통신 프로토콜

WebImg 시스템 구성요소 JR, JP, JM들의 연결에서부터 이들간의 상호작용을 통해 전달되는 정보는 관련된 테이블에 기록되며 기록된 정보를 통해 요소들간의 작업 수행 상태를 판단한다. 그림 2는 구성요소들간의 WebImg 시스템 개시부터 연산하고 종료까지의 통신 프로토콜이다.

주어진 애플리케이션에 대한 작업을 기대하는 JR과 작업에 참여하기를 원하는 JP는 사전에 JM를 방문함으로써 수행을 위한 준비작업으로 JR과 JP의 애플릿을 HTTP 서버로부터 업로드한 다음(①), 업로드된 애플릿의 URL을 JM에게 등록한다(②). 특히, JP는 등록과 함께 수행능력을 테스트하는 간단한 평가 연산[12]를 하여 시스템 초기 정보를 JM에게 알린다. 동시에 JP 시스템에 참여할 준비를 알리는 JPR 메시지를 전송하고 JM의 응답을 기다린다.

URL 등록 인증으로 참여한 JR과 JP는 식별자를 부여받고(③), JR은 준비단계를 알리는 JRR 메시지

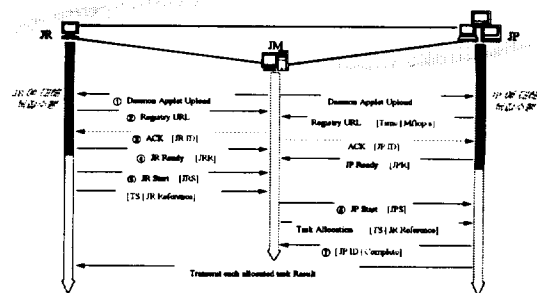



그림 2. WebImg 통신 프로토콜

를 전송하여 JM에게 알린다(④). JR은 병렬 처리를 수행할 애플리케이션을 가지고 있으며 작업 시작을 알리는 JPS 메시지와 해당 작업 스펙(TS : Task Specification)을 통신 오버헤드로 인한 전체 수행시간에 드는 비용을 줄이기 위한 메커니즘으로 JP의 결과를 받기 위해 원격으로 참조가 가능하도록 JR의 정보인 작업 요청자 참조값(JRReference)과 보낸다(⑤). 요청을 받은 JM은 JP에게 작업을 할당하게 되는데, JP는 해당하는 작업 스펙과 함께 JR로부터 보내져온 JRReference를 받는다(⑥). 이때, 작업 스펙으로는 JP 등록과 함께 수행능력을 테스트한 정보를 성능비대로 스케줄링 하여 해당 JP에게 작업을 할당하고 참여한 모든 JP가 같은 시간에 작업을 마치기 위한 작업 할당 전략에 의해 해당하는 TS이 정의된다. 연산을 마친 JP는 자신의 결과를 독립적으로 식별자와 함께 Complete 메시지를 전송하며 JR이 작업을 요청하면서 전달했던 JRReference에 의해 JM을 거치지 않고 직접 JR에게 응답에 대한 결과를 보냄으로써 애플리케이션 수행을 마친다(⑦).

3. 유연성 있는 작업 할당 기법

인터넷 환경에서는 혼합 이기종 시스템의 사용으로 인하여 그들의 수행능력은 다양하다. 이로 인해 요청된 애플리케이션에 대해 다수의 JP에게 수행능력에 따라 작업을 분배하여 같은 시간에 작업의 완료가 이루어지도록 한다. 하지만 웹이라는 물리적인 거리와 가변적인 환경에서 여러 일반 사용자들에게 마저 개방되어 있으므로 이들 시스템 가용성의 정도에 따른 참여자들의 컴퓨팅 사이클을 방해한다. 초기 성능 정보만을 가지고 작업 할당 과정을 거치고 수행이 완료되기를 기다린다는 것은 수시로 변하는 인터넷의 특성에서 정확성이 있으면서 최상의 수행 결과에 이르기는 힘들다. 이러한 불규칙적인 변화, 즉 수행 상태와 호스트의 고장에 대응할 수 있는 컴퓨팅 알고리즘이 필요하며, 최적의 수행시간을 얻기 위한 분석을 제안하게 되는데 본 논문에서는 유연성 있는 작업 할당(Flexible Task Allocation)을 제시한다.

먼저 JP는 자신을 병렬 플랫폼에 등록하면서 균형적인 작업을 하기 위한 과정을 거친다(❶). 작업이 시작되기 전에 성능을 고려하여 적절한 분배가 이루어지도록 컴퓨터의 연산능력을 평가해 오던 간단하면서도 신뢰성을 확보할 수 있는 평가를 거친다. 이것으로는 LINPACK 벤치마크[12]를 사용한다. 가우스 소거법(Gaussian Elimination)의 부분 피벗팅 기법을 이용해 행렬에 대한 선형방정식 ($Ax = B$)의 해를 구하는 연산을 한다. 결과로 단위 시간당 부동 소수점 덧셈과 곱셈 연산의 수행 회수를 측정하여 초당 몇 백만 번의 부동 소수점 연산을 수행했는지에 대한 MFLOPS (mega-floating point operation per second) 단위 값을 제공하며 측정된 값에 비례하여 JP에게 초기의 수행 작업 양을 결정한다. 벤치마크 정보는 JM에 의해 유지된다.

JR의 작업 시작과 함께 컨트롤러에서 계산 단위라고 정의한 2^k형태의 컴퓨니트(Computation Unit : compunit)를 전송한다(❷). 참여한 JP들은 성능비에 따라 할당받은 작업 양과 비례하여 각각 다른 컴퓨니트 수를 가지게 된다. 컴퓨니트 수에 비례하여 계산된 전체 작업량을 JR에 의해 작업의 시작을 알리는 메시지를 받는 순간 모든 JP에게 작업을 브로드캐스트하며 작업 수만큼 쓰레드를 생성하며 작업간에 독립적으로 연산을 수행한다. 요구된 전체 작업량 *SizeTotalJob*에서 컴퓨니트 크기가 *SizeCU*이며, 동일한 시간에 벤치마크를 수행한 Mflop/s값의 집합을 $\{P_0, P_1, P_2, \ldots, P_{NumJP-1}\}$라 할 수 있으므로 임의의 JP에 대한 성능비를 고려해 할당된 작업량은 다음 식 (1)로 정의한다.

$$TaskJP_i = \frac{P_i}{\sum_{j=0}^{NumJP-1} P_j} \cdot \frac{SizeTotalJob}{SizeCU} \quad (1)$$

식 (1)을 살펴보면, 임의의 JP가 할당받은 작업량은 계산 단위 회수인 컴퓨니트 수와 비례한다. 할당받은 작업을 연산을 하는데 그 수행 상태를 동시에 컴퓨니트 단위로 자바 소켓을 통해 JM의 서브관리자인 JO에게 수행 정보를 전송하고 RMI를 이용하여 응답에 대한 결과를 JR에게 전달한다(❸). 결과에 대한 양방향 응답은 할당받은 작업의 처리 도중 발생 가능성에 대한 검사(check)의 포함이다. 발생 가능성으로는 초기의 할당된 연산에 대한 커다란 변화를 가져올 가능성과 연산 도중 호스트의 고장으로 작업에 대한 응답을 하지 못할 경우로 한다. 환경의 가변적인 상황에 따라 작업 수행 능력의 변화, 참여 호스트의 고장에 대해 유연성을 적용하여 JP 고장 발생 유/무로 가정하여 해결책을 찾는다.

3.1 작업 처리자중 고장이 없다고 가정

본 시스템은 참여한 JP수에 대한 제한이 없음을 가정한다. 그림 3은 JP중 고장이 발생하지 않았을 경우의 유연성 있는 작업 할당을 보여준다.

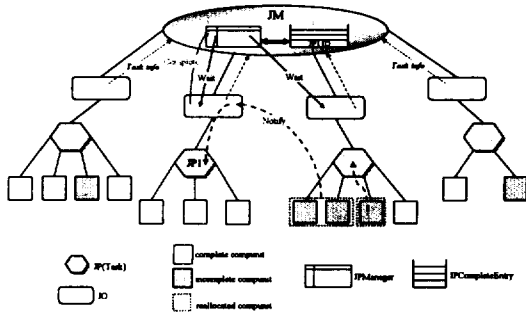



그림 3. 작업 처리자중 고장이 없다고 가정한 경우

식 (1)의 연산에 의해 성능비대로 할당받은 작업들을 수행하는 다수의 JP를 가지며, JM측에는 JP의 식별키를 매핑하기 위한 Hashtable인 JPManager와 할당받은 작업의 연산을 마친 JP를 기록하기 위한 JPCompleteEntry가 있다. 실질적인 JP 관리는 JP마다 생성된 쓰레드로 JO에 의해 이루어진다. 초기에 할당받은 작업 정보를 가지고 수행하면서, 컴퓨니트 단위로 작업 진행 상황을 JO에 기록한다(❶).

JO가 가지고 있는 정보에 의해 할당된 작업 처리율(processing ratio)에 따라 우선순위(priority)를 얻는다(❺). 할당받는 작업에 대해 처리 속도가 빠른 JP에게는 높은 우선순위 ($priority(NumJP-1)$)를 부여하고 늦은 처리 속도의 JP에게는 낮은 우선순위 ($priority0$)가 부여된다.

임의의 JP가 할당받은 작업을 마쳐서 JO를 거쳐 작업 관리자에게 Complete 메시지를 보내면 일단 JP 식별자는 JPCompleteEntry 큐에 저장되고 $priority(NumJP-1)$가 주어지고 Wait 메시지를 받는다. 이때의 Wait 메시지는 작업 연산은 하지 않으면서 다른 JP에 의해 미수행된 작업을 재할당 받기 위한 JM의 스케줄링을 기다린다.

작업을 수행하기 위해 생성되었던 쓰레드는 stop() 메소드를 호출하지 않고 쓰레드 재사용(ReThread)한다. Notify 메시지와 함께 재할당 연산에 참여할 때 많은 비용이 드는 쓰레드 생성 시간을 줄일 수 있기 때문에 전체 수행시간을 줄인다. Complete 메시지를 보낸 JP를 제외하고 모든 JO에게 지금까지 할당받아서 연산을 마친 컴퓨니트 수를 결과로써 알린다. 결과는 JO에 의해 전송되며 작업 처리율을 적용하여 재할당(ReAllocation) 연산 여부를 예측하며, 이때 JP들은 계속해서 작업을 수행한다.

한편, JO가 전송한 결과를 예측하여 낮은 우선순위 JP에게 작업을 일시 중지하기 위한 Wait 메시지를 전송하여 작업 정보에 따라 재할당 연산을 수행한다. 낮은 우선순위의 JP가 아직 수행되지 못한 작업을 처리하기 위해 할당된 작업을 끝낸 높은 순위 JP와 처리율에 비례하여 작업을 재할당한다(❻).

어떤 JP_k 가 할당받은 컴퓨니트 수인 $NumCuJP_k$의 연산을 모두 마쳤을 경우 이 때, 낮은 우선순위 JP_j가 할당받았던 컴퓨니트 수 $NumCuJP_j$ 중에 미수행된 컴퓨니트 수가 $NuminCuJP_j$ 라면, JP_k가 재할당 받을 작업 양에 비례하는 컴퓨니트 수는 식 (2)와 같다.

$$NumReCuJP_k = \frac{NumCuJP_k \cdot NuminCuJP_j}{NumCuJP_k + (NumCuJP_j - NuminCuJP_j)} \tag{2}$$

재할당 연산에 참여한 JP를 제외하고는 계속해서 작업을 하며 JO에 기록된다. 재할당 연산을 통해 초기 할당받은 작업의 연산을 마친 성능이 좋은 JP도 다시 작업에 참여하게 된다. 가변적인 상황에 따라 작업 처리 속도에 차이가 나게 되어 유휴 상태의 JP가 발생되지 않고 동일한 시간에 작업을 종료한다. 요구된 작업 결과가 종료되었음을 알리는 JM이 모든 참여 JP에게 Stop 메시지를 보낼 때까지 (❻)의 과정을 반복한다.

할당받은 작업에 대해 작업 처리율이 100%에 가까워질수록 JM과 JP의 연산시간에 비해 이들간의 통신시간이 증가하여 전체 수행시간이 커질 수 있다. 그러므로, complete의 상한을 규칙적으로 한정하기 위한 레벨(level)을 적용한다(❼). 레벨은 작업 처리율에 기반을 두며 6단계(40%~90%)로 정의된다.

JPCompleteEntry 큐에 삽입된 할당·재할당 받은 작업을 마친 JP는 JM의 스케줄링을 기다리는데 작업 처리율이 설정된 레벨에 아직 미치지 못한 JP만이 재할당 연산에 참여한다. 설정한 레벨에 이미 도달한 작업 처리율을 가진 JP는 JM에 의한 재할당 연산에 참여되지 않고 미수행된 컴퓨니트 연산을 계속하며 작업 종료 후 Complete 메시지를 전송한다.

이러한 재할당 연산은 다음 표1에서 정의한 파라미터를 이용하며, JP중 고장이 없을 경우 유연성 있는 작업 할당 알고리즘 그림4를 따른다. JP가 JO에게 작업 수행 상태를 전송하면서 JR에게 응답의 결과를 전송한다(❽).

```
allocate ∀TaskJP_i to ∀JP_i ;
do {
    receive(Complete) from ∃JP_i ;
    add(JP_i ID);          // add the completed JP to JPComplete Entry Queue
    JPCC ++;
    send(Wait) to JP_i ;        // where JP_i = JP_C
    // search low - prio rity JP_j from JPManager, where JP_C ≠ JP_j
    if ((JP_jRatio < level) && (priority == priority0)) {
        send(Wait) to JP_j ;
        reallocate inCuJP_w , JP_C and JP_w respective ly ;
        delete(JP_C ID); // delete the reallocate d JP from JPComplete Entry Queue
        JPCC -- ;
        reset priority of reJP ;
    }
    else {
      compute the allocated Task continuous ly ;
    }
} while(JPCC == NumJP);
```

그림 4. JP중 고장이 발생하지 않는 경우의 알고리즘

표 1. 유연성 있는 작업 할당 알고리즘에 적용한 파라미터

파라미터	의 미
JP	$JP_1, JP_2, \cdots, JP_{NumJP-1}$를 이루는 전체 작업 처리자
$TaskJP$	$SizeTotalJob = \sum_{i=0}^{NumJP-1} TaskJP_i$, $TaskJP_i$는 임의의 JP_i가 할당받은 작업의 양 즉, $JP_1, JP_2, \cdots, JP_{NumJP-1}$가 각각 할당받은 작업
$level$	사용자가 정의한 complete의 한정값
JP_{jRatio}	작업 처리자 j의 작업 처리율
JP_C	Complete 메시지를 보낸 우선순위가 높은 작업 처리자
JP_W	Wait 메시지를 받은 우선순위가 낮은 작업 처리자
JP_F	고장이 발생한 작업 처리자
$reJP$	재할당 연산에 참여하는 작업 처리자
$JPCC$	수행을 마친 작업 처리자의 수
$NumCu$	수행된 컴퓨니트 수
$NuminCu$	미수행된 컴퓨니트 수
$inCuJP_W$	Wait 메시지를 받은 작업 처리자의 미수행된 컴퓨니트
$inCuJP_F$	고장이 발생한 작업 처리자의 미수행된 컴퓨니트

3.2 작업 처리자중 고장이 발생한다고 가정

WebImg는 모든 참여자들이 웹 환경에 브라우저를 통하여 연결된다. 그러므로 시스템을 수행하는 도중에 참여 호스트와 네트워크에서의 결함 발생 가능성이 높다. JM이 할당한 작업을 JP가 처리하는 동안 네트워크 결함으로 인하여 JR이 응답을 받지 못하는 상황을 고려한다. 그림 5는 JP의 고장이 발생한 경우의 유연성 있는 작업 할당 기법이다.

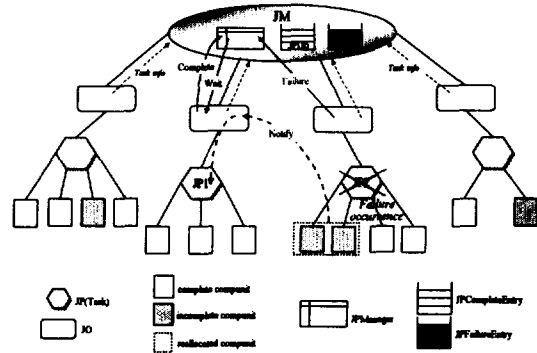



그림 5. 작업 처리자중 고장이 발생한다고 가정한 경우

JP가 할당받은 작업을 컴퓨니트 단위로 처리하여 연산 정보를 JO에 기록하며, JPManager에서 모든 JO 정보를 얻는다. 그러므로, JM은 참여한 모든 JP의 작업 처리 진행 상태를 안다. 또한 전송되는 과정에서 자바가 지원하는 소켓관련 예외 처리를 적용하여 JP 고장을 탐지한다.

고장이 탐지되면 Failure 메시지가 전송되고 JM은 먼저 JPFailureEntry 스택에 고장난 JP 식별자와 미수행된 컴퓨니트 수가 삽입된다. 고장이 발생한 JP를 식별하기 위해 객체의 스택을 캡슐화한 것으로 Vector 클래스를 상속받았으며 push() 메소드를 사용하여 저장된다. 고장이 발생한 JP의 미수행된 컴퓨니트는 JPCompleteEntry에 저장되어 있는 JP가 전담 수행한다.

JPFailureEntry에 삽입된 JP에게는 낮은 우선순위를 부여하고 JPCompleteEntry에 있는 높은 우선순위 JP에 의해 수행된다. 만약에, JPFailureEntry가 비어있으면 고장이 없다고 가정한 경우와 동일하게 수행한다.

고장이 발생하여 일단 JPFailureEntry에 추가된 JP는 더 이상 서비스에 참여할 수 없으며 작업 정보는 통신시간을 고려하여 한정을 둔 레벨을 따르지 않으며 무조건 미수행된 컴퓨니트는 JPComplete-

Entry에 저장된 JP에 재할당되어 수행된다. 그림 6은 JP중 고장이 발생했을 경우의 작업 할당 알고리즘이다.

```
JPFailureEntry = Null;
do {
    if(∃ JP_i is Failure){    // JP_i = JP_F
        push(JP_F.ID);    // add the failure - occurrenced JP to JPFailureEntry Stack
        give low-priority to JP_F ;
        JPCC++;
        if(JPCompleteEntry != Null){
            reallocation inCuJP_F to JP_C ;
            delete(JP_C.ID);  // delete the reallocated JP from JPCompleteEntry Queue
        }
    }
    else{    // receive(Complete) from ∃ JP_i
        if(JPFailureEntry!= Null){
            reallocation inCuJP_F to JP_C ;
            pop(JP_F.ID);  // delete the reallocated JP from JPFailureEntry Stack
        }
        else{
          // compute the case that failure has not happened ;
    }
} while(JPCC == NumJP);
```

그림 6. JP중 고장이 발생하는 경우의 알고리즘

3.3 수행시간 분석

애플리케이션을 처리하는데 작업 요청에서부터 결과에 대한 응답을 하는데 까지 전체 수행시간은 연산시간과 통신시간의 합으로 정의된다. 연산시간은 JR이 요청한 작업을 JP에게 할당하기 위한 JM의 스케쥴링시간 (*timeJM*)과 할당받은 작업을 처리하는데 걸리는 JP의 수행시간 (*timeJP*)을 포함한다. 통신시간은 JR에 의해 작업을 JM에게 요청하는데 걸리는 등록시간 (*timeJRM*)과 연산을 마친 JM은 성능비대로 스케쥴링된 결과를 전송하는데 걸리는 할당시간 (*timeJMP*)을 JP의 작업 상태를 제어할 목적으로 할당받은 작업 수행 상태를 JM에게 알리는 검사시간 (*timeJPM*), JR에게 결과를 응답으로 이미지를 그리는 시간 (*timeJPR*)으로 구분한다.

그렇다면 초기 벤치마크 연산을 수행한 JP의 성능평가 정보에 의존한 전체 수행시간을 구하면 다음 식 (3)과 같다.

$$T_{total} = timeJRM + timeJM_0 + \max_i \{ JP_i^{'} \mid 0 \leq i \leq (NumJP-1) \} \quad (3)$$

식 (3)을 살펴보면, JP가 JM에 성능평가 정보를 이용해 식 (1)에 의해 각각의 작업 양을 결정하는 연산시간이 $timeJM_0$이며, 할당된 작업간에는 의존관계가 없다고 가정했으므로 JP와 관련된 수행시간은 JM의 스케쥴링을 통해 JP에게 해당 작업을 할당하는 통신시간과 각각 JP에게 할당받은 작업을 처리하는 연산시간과 연산한 결과를 응답하는 시간을 포함하는 최대값으로 한다. 그러면,

$T_{JP(i)'} = timeJMP_i + timeJP_i + timeJP_iR$ 이다.

$\max_i \{T_{JP_i}^{'} \mid 0 \leq i \leq (NumJP-1)\}$은 식 (3)의 전체 수행시간을 결정하는 요인이라고 볼 수 있다. 하지만 JM에 의해 단 한번 초기 성능비 연산과정을 거쳐 JR은 JP들의 작업 수행에 대한 응답을 기다린다. 즉, *timeJPM* 시간은 배제한 것으로 만약에 환경적 요인으로 어떠한 JP의 극심한 작업량 증가로 성능비에 의해 할당된 연산을 제대로 수행하지 못하여 전체의 수행시간이 커지거나 또는 JP의 고장으로부터 JR에게 응답이 가지 않을 경우에는 전체 수행시간을 결정하는 JP가 무한정 ($T_{JP(i)}^{'} \rightarrow \infty$)되므로 전체 수행시간을 알 수 없다. 극심한 성능변화와 무한정 기다릴 가능성을 예측하여 유연성 있는 작업 할당을 통한 JP들의 수행 변화에 따른 제어를 통해 재할당 연산을 수행한다.

유연성 있는 작업 할당에서 작업 상태를 알리는 JM과 JP간의 메시지 전송이 이루어지는데 Complete 메시지를 받은 회수를 c라 가정한다. 재할당 연산에 참여한 JP의 성능비대로 재할당 작업의 양을 결정하는 식 (2)를 적용하며 이러한 재할당 연산시간을 $timeJM_{re}$ 이라 하면 다음 식 (4)를 얻는다.

$$T_{total} = timeJRM + timeJM_0 + (\mathrm{c} - \mathrm{NumJP})\, timeJM_{re} + \max_i \{ T_{JP_i} \mid 0 \leq i \leq (NumJP-1) \}$$

여기서, $T_{JP(i)} = timeJP_MM + FtimeJP_i$

재할당 연산 회수는 할당된 작업을 마친 모든 JP는 Complete 메시지를 전송하므로 Complete 메시지 회수와 참여 JP수의 차에 의해 결정된다.

$\max_i \{ T_{JP_i} \mid 0 \leq i \leq (NumJP-1) \}$는 작업간의 낮은 상관 관계에 의해 JP측에 해당하는 수행시간으로 병렬 수행의 최적시간을 결정하는 요인이다.

그림 3과 같이 JP와 JM간의 작업 수행 응답의 메시지를 전달하는 JO가 있다. JO와 JM의 메시지 전송시간은 JP의 연산시간에 동시에 이루어진다. 초기 할당받은 작업과 재할당의 연산을 통해 할당받은 작업에 대해 완료했을 경우에 응답으로 Complete 메시지 전송시간과 재할당 연산 수행 시 또 다른 쓰레드 생

성시간을 줄이기 위한 쓰레드를 종료하지 않고 대기하라는 Wait 메시지 전송시간과 다른 한편으로 할당받은 작업의 처리 속도가 느린 원인으로 미수행된 컴퓨니트가 많을 경우 일단 작업 처리를 중지하고 재할당을 기다리는 Wait 메시지를 전송하는 등의 통신시간이 추가되는데 식 (4)에서는 이러한 메시지 전송시간을 $timeJP_{M}M$ 한다. 해당 JP의 수행능력에 따라 초기에 할당받은 작업과 재할당받은 작업에 대한 연산을 마친 컴퓨니트 총 수를 $NumTotalCu$ 라하고, 연산 결과의 응답을 동시에 JO를 통해 JM과 JR에게 전송하므로 $\max\{timePMCu_j, timePRCu_j\}$이다. 그러면, 임의의 JP가 유연성 있는 작업 할당에 참여한 수행시간 $FtimeJP_i$ 는 컴퓨니트 단위의 연산과 통신에 의해 $\sum_{j=1}^{NumTotalCu}\{FtimeJP_iCu_j + \max\{timePMCu_j, timePRCu_j\}\}$으로 정의된다.

JP들의 극심한 수행 능력의 변화에 대응하기 위한 많은 메시지 전송이 JM과 JP간에 이루어진다. 메시지 전송에 의해 연산 정보를 통해 JP의 수행 상태를 얻어 작업 처리율을 가져와 재할당 연산을 수행한다. 메시지 전송을 위한 통신시간 및 JM의 재할당 연산시간이 성능에 변화가 거의 없이 일정하게 유지될 경우에는 많은 통신시간과 연산시간의 추가로 전체 수행시간이 클 수 있지만 서비스가 시작된 후로 참여한 JP들이 초기의 성능평가 정보와는 다른 성능변화가 발생할 경우 제안한 유연성 있는 작업 할당을 통한 전체의 수행시간을 줄여 최적화 비용을 얻는다.

4. WebImg 구현 및 적용사례

구현을 위해 자바 JDK(Java Development Kit) 1.2와 사용자 인터페이스를 위한 자바의 경량 컴포넌트(light-weight Component)인 Swing을 이용하였다. 웹 브라우저는 Internet Explorer 5.0을 이용하였으며 호스트간의 통신은 자바 소켓과 원격 메소드 호출을 사용하였다.

4.1 WebImg 시스템 구성도

그림 7은 가상 병렬처리 시스템인 WebImg의 구성 요소별 관계를 나타내는 블럭 다이어그램이다. JR측은 작업을 요청하고 결과를 얻는 작업 사용자 인터페이스(Task User Interface)와 전달된 액션 처리를 하는 작업 엔진(Task Engine)으로 구분한다. UI는 JR을 다루는 작업 엔진에 의해 JM세션을 이용하여 작업을 알린다. JP측은 JM세션을 가지고 작업 엔진에서 연산 결과를 얻는다. JP의 작업 엔진은 할당 · 재할당 받은 태스크에 대한 연산을 수행하며 식별되는 세션을 통해 연산 정보를 JO에게 보내며 동시에 JR에게는 JM을 거치지 않고 JR의 작업 UI에 직접 전달된다. 한편, 작업 엔진은 JP마다 작업 처리를 위해 식별할 수 있는 세션 정보에 의해 연산에 차이를 가져온다. 등록된 JR과 JP를 관리하는 JM측은 자바가 제공하는 소켓과 서버소켓에 의해 얻어지는 입력과 출력은 세션을 통하여 정보 데이터를 전송한다. 세션에 의해 연결된 구성 요소들을 이용해 스케줄링을 한다. JO와 세션 연결을 통해 해당 JP의 연산 상태와 작업 스케줄링 상태를 관찰한다.

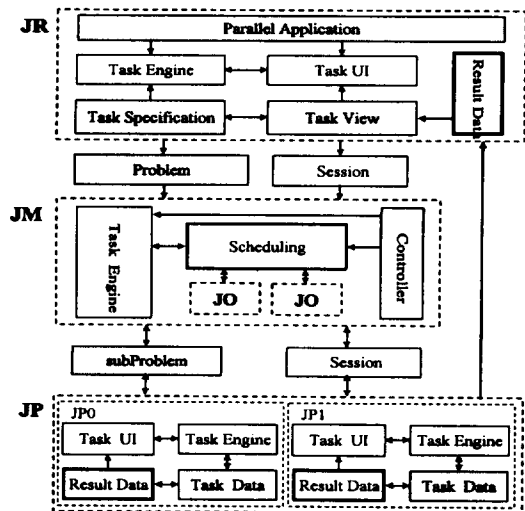
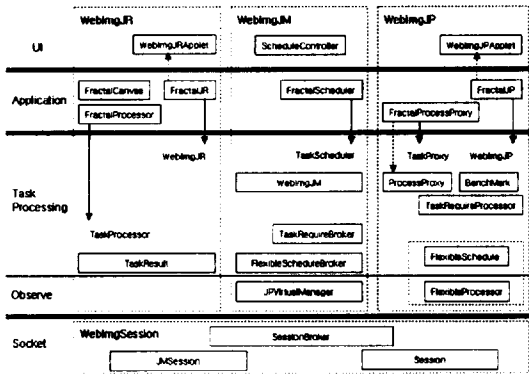



그림 7. WebImg의 블럭 다이어그램

4.2 WebImg 클래스 구조

다음 그림 8은 WebImg 시스템의 자바 클래스 구조이다. 맨 하부에 자바가 지원하는 소켓을 통해 구성요소들의 입력과 출력에 대한 인터페이스를 제공하며 이들의 클래스를 하나의 WebImgSession로 패키지화하여 공통 연결 통로를 설정한다. WebImg-Session상에 JR, JM, JP는 각각 사용자 인터페이스층(UI layer), 애플리케이션층(Application layer)과 애플리케이션에 대해 병렬 수행하는 작업처리층(Task Processing layer)을 이룬다.

그림 8. WebImg 클래스 구조

그림 10. 재할당 연산 수행 모습

4.3 적용사례

본 논문에서 적용되는 애플리케이션으로 간단한 알고리즘을 사용하면서 통신시간에 비해 처리를 위한 연산시간이 매우 큰 프랙탈 이미지 처리이다. 아주 간단한 복소수 변환 ($z=z^2+c$ $(z=x+yi,$ $c=c_1+c_2 i))$을 계속 반복하여 계산하면 복잡한 구조를 만들어 내는 만델브로트(Mandelbrot) 방법을 사용한다. 알고리즘 내부에는 자기 유사성(self-similarity)을 전제로 하여 끊임없는 자기 복제(recursiveness)를 반복하므로 하나의 이미지를 생성하는데 상당히 많은 수행시간과 처리량이 요구된다.

그림 9는 작업 요청자 부분으로 이미지 처리 결과를 얻기 위한 뷰, 애플리케이션의 다양성을 위한 스펙, 상태를 탐지하기 위한 상태영역을 가지며 결과를 얻는 화면이다. 컴퓨니트 크기를 2, JP 4대로 고정하여 연산 수행 중 급격한 JP1의 성능 저하로 재할당 연산을 수행하는 모습을 그림 10에서 보인다.

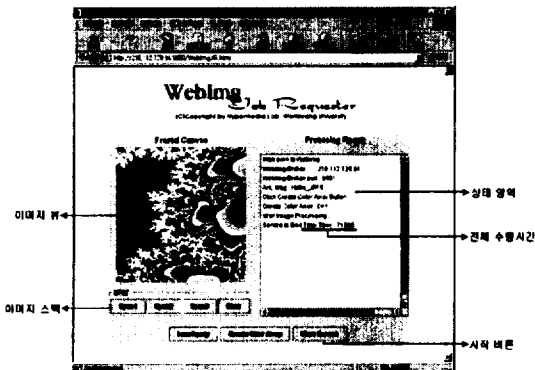



그림 9. 작업 요청자 실행 화면

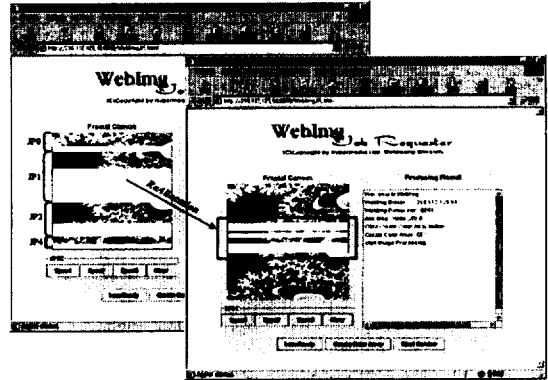
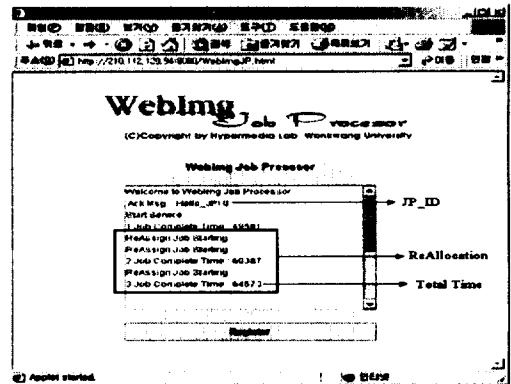



그림 11. 작업 처리자 실행 화면

3번의 재할당 연산을 수행하는 작업 처리자의 실행 모습은 그림 11이다. 시스템 내부의 처리 정보를 볼 수 있는 작업 관리자의 데몬으로 그림 12이다. 유연성 있는 작업 할당 전략을 따르며 1대의 JP고장이

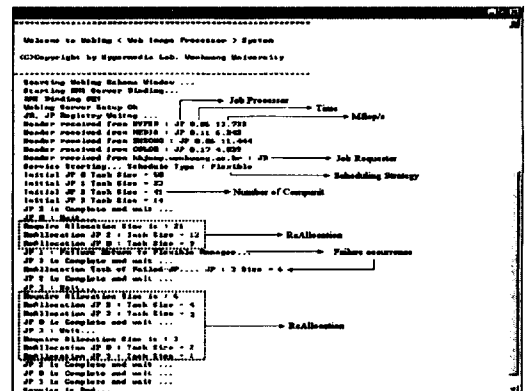



그림 12. 작업 관리자 데몬

일어났으며 3번의 재할당 연산을 거쳐 작업을 마친 상태이다.

4.4 성능평가

컴퓨팅 환경으로는 네트워크로 연결된 이종의 10대의 컴퓨터를 이용한다. 자바 수행 능력을 가진 Netscape 4.0 이상 또는 Explorer 5.0 이상을 탑재하고 있는 SUN SparcStation 20과 Pentium PC로 구성된다. Pentium PC들은 다양한 운영체제, CPU, 메모리를 가지며 세부사항은 표 2와 같다.

표 2. 이기종 시스템의 구성

기능	식별자	OS	CPU	RAM
JM	JM	Solaris2.6	Sun Sparc20	64MB
JR	JR0	WindowsNT	PentiumIII 500MHz	256MB
JP	JP0	Windows2000	PentiumIII 550MHz	64MB
	JP1	Windows98	PentiumIII 500MHz	128MB
	JP2	Windows98	PentiumII 233MHz	32MB
	JP3	Windows95	Pentium 150MHz	32MB
	JP4	Windows2000	PentiumIII 550MHz	256MB
	JP5	Linux5.2	PentiumIII 500MHz	128MB
	JP6	WindowsNT	PentiumIII 500MHz	128MB
	JP7	Windows95	Pentium 150MHz	16MB

4.4.1 JP수 증가에 따른 성능평가

작업 할당 전략으로는 참여 JP의 수에 비례하는 균등하게 작업을 할당한 단순 할당, JP가 가지는 초기 성능평가 정보에 따라 할당된 정적 할당, 가변적인 환경에 적응할 수 있는 유연성 있는 할당을 비교한다.

단순 할당 경우 성능이 낮은 JP에 의해 수행시간의 증가로 8대로 실험한 결과 스피드업이 4를 넘지 못하는 매우 낮은 효율성을 보인다. 하지만 성능의 변화가 많지 않은 경우일지라도 정적 또는 유연성 있는 할당이 JP수의 증가에 따라 스피드업이 6.5를 넘는 비교적 선형 그래프를 이룬다. JP수 추가로 점점 선형도는 떨어지며, 이러한 그래프는 애플리케이션의 종류에 따라 다소 차이를 가지지만 본 논문에서 적용된 프랙탈 이미지 처리에서 JP수가 7에 이르면서 스피드업 증가도가 감소하다가 8대를 넘어서면 거의 증가하지 않아 더 빠르게 실행할 수 없다. JP수에 비례하여 스피드업 결과를 얻지 못하는 것은 알고리즘에의 통신비용과 동기화 오버헤드가 일어나 손실이 JP수를 추가함으로써 일어나는 이익만큼 커지기 때문이다.

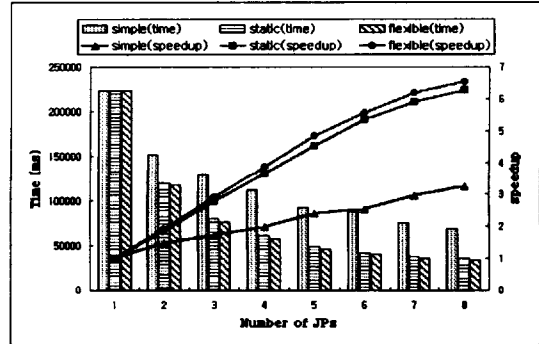



그림 13. JP수 증가에 따른 성능평가

4.4.2 유연성 있는 작업 할당의 성능향상

이기종 시스템으로 구성된 표 2의 작업 처리자중 JP0~JP3인 4대의 시스템을 이용하며, 계산 단위인 컴퓨니트 크기를 2로 고정하여 실험한다. 뿐만 아니라 글로벌 컴퓨팅을 적용하기 위해 작업 처리자중 JP2는 학교망을 벗어나 지리적으로 떨어진 곳에 위치하며 작업 요청자와 작업 관리자를 분리하여 각각 다른 연구실에 배치하여 평가한다.

표 3. 할당받은 컴퓨니트 수 비교

작업 처리자	Mflop/s	성능비	단순 할당	정적 할당	유연성 있는 할당: 성능변화가 거의 없는 경우	유연성 있는 할당: 성능변화가 심한 경우
JP0	13.733	1.00	32	50	47	61
JP1	11.444	0.83	32	41	41	38
JP2	6.242	0.45	32	23	26	12
JP3	4.039	0.29	32	14	14	17

제안한 작업 할당 전략으로 JP중 고장이 발생하지 않는다고 가정할 때 작업 할당 전략에 따라 표3과 같은 컴퓨니트 수를 이룬다. 유연성 있는 할당의 레벨이 80%로 설정하였을 경우 성능의 변화가 거의 없는 경우에도 1~2번의 재할당 연산을 수행한 결과이다. 한편, 수행 도중 JP2와 JP1의 성능변화가 극심하여 할당받은 작업을 미쳐 수행하지 못하여 4번에

서 많게는 7번의 재할당 연산을 수행하여 정적 할당과는 컴퓨니트 수가 많은 차이를 가진다.

그림 14는 유연성 있는 작업 할당의 성능 향상의 실험이다. JP중 고장이 없을 때 성능변화가 없는 경우에는 단순 할당에 비해 정적 할당은 50.41%의 성능향상을 가져와 전체 수행시간의 큰 차이를 주며 정적 할당에 비해 유연성 있는 할당의 경우 4%내외의 차이가 있다. 이처럼 성능변화가 거의 없는 경우는 정적 할당의 벤치마크에 의한 성능평가가 다소 신뢰성을 가져온다. 하지만, 성능 변화가 심한 경우에는 유연성 있는 작업 할당과 정적 할당은 36.84% 성능 차이를 가진다. 웹의 가변적인 원인으로 벤치마크한 CPU의 능력이 전체 수행시간에 완전한 영향을 주지 못한다. 더욱이 수행 도중 JP의 고장 발생의 경우에는 정적 할당의 경우에는 고장이 발생하면 전체 수행시간은 무한대가 되어 JR은 응답을 받을 수가 없다. 그러므로 제안한 유연성 있는 할당에서는 고장난 JP가 아직 끝내지 못한 작업을 다른 JP가 대신하여 전체 수행시간을 얻으며 또한 JR은 요청한 작업의 결과를 얻는다. 하지만 고장이 발생한 JP는 더 이상 연산을 하지 않으므로 연산을 수행하는 JP수가 감소하며 미수행된 작업을 가져오기 위한 재할당 연산의 추가로 전체 수행시간이 증가한다.

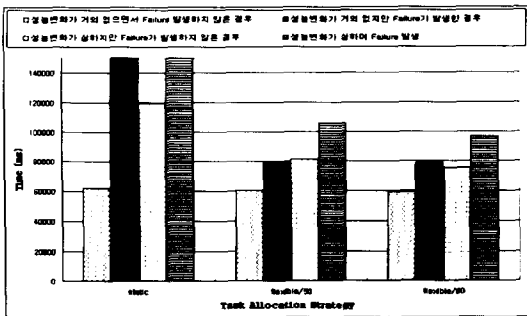



그림 14. JP중 고장 발생의 유/무에 따른 평가

4.4.3 컴퓨니트에 따른 평가

WebImg 시스템 JM측 컨트롤러의 계산 단위인 컴퓨니트 설정에 따른 실험이다. 그림 15는 성능변화가 거의 없는 경우 컴퓨니트 크기의 변화에 따른 연산시간과 통신시간의 변화이다. 작업 양은 컴퓨니트 크기에 의해 할당받는 컴퓨니트 수를 가지며 JP는 컴퓨니트 단위로 연산을 수행하여 결과에 응답한다. 그러므로 컴퓨니트 크기가 클수록 알고리즘 내의 연산 회수의 감소로 연산시간과 통신시간이 줄어 전체 수행시간이 떨어진다.

하지만 성능변화가 극심하여 재할당 연산이 일어나는 경우는 그림 16과 같이 컴퓨니트 크기의 증가에 따라 전체 수행시간의 증가로 성능이 저하된다. 이것은 JM이 컴퓨니트 단위로 수행 상태를 탐지하므로 컴퓨니트 크기가 클수록 연산 회수가 적어서 통신시간은 감소하나 연산 도중에 수행 중인 컴퓨니트를 중지하고 재할당이 이루어지므로 일부분 연산되고 있는 컴퓨니트 연산시간의 손실을 가져와 전체 수행시간이 크다.

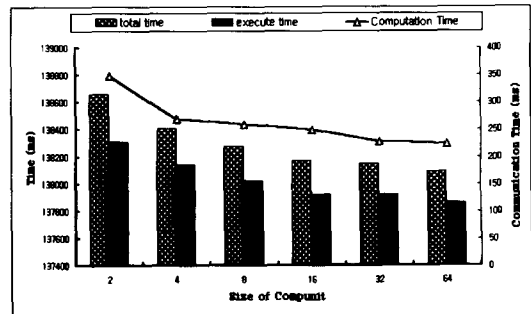



그림 15. 컴퓨니트 크기에 따른 성능평가

5. 결 론

웹에 연결된 수많은 컴퓨터를 통합하여 하나의 거대한 가상 시스템이 이루어지는데 이들 안에 유휴 자원을 활용함으로써 고성능을 필요로 하는 애플리케이션을 병렬 수행함으로써 비용은 저렴하면서 성능면에서도 월등한 계산 환경을 마련하기 위한 가상 병렬 처리 시스템인 WebImg를 설계 · 구현하였다. WebImg 시스템은 웹 환경에서 거대 작업을 여러 개의 서브 작업으로 나누어 병렬 처리함으로써 처리속도를 향상시킨 작업 할당 전략을 제의한 것이다.

예측할 수 없는 작업 처리 시스템들의 가용성 및 상태 즉, 호스트의 고장(failure)에 대해 유연성 있게 대처하므로 결함내성을 제공한다. 시스템의 수행능력의 변화에 우선순위를 부여하여 재할당 연산을 진행하며 재할당을 제한하는 레벨을 제시하여 처리시간의 최적화를 얻었다. 인터넷에 수많은 호스트들에게 작업을 분배하고 그 결과를 전달하는데 있어서

작업 관리자의 중재없이 애플리케이션의 수행을 요구한 작업 요청자에게 직접 전달하는 애플릿간의 통신이 가능하도록 원격 객체 참조 모델을 사용하였다. 그러므로 작업 관리자의 부담을 줄이고 통신 오버헤드를 방지하였으며 전체 수행시간을 줄였다. 뿐만 아니라 작업 관리자의 중재가 없는 대신에 서브관리자인 작업 처리자의 가상 쓰레드인 작업 관찰자를 둠으로써 응답에 대한 신뢰성이 있다.

본 논문은 이러한 유연성 있는 작업 할당을 평가하기 위해 WebImg 시스템의 활용으로 통신시간에 비해 많은 계산량을 가진 작업들을 이룰 수 있는 프랙탈 이미지 처리를 이용했다. 작업 처리자수 증가가 8대에 이를 때까지는 거의 선형 그래프를 이루는 성능향상을 얻었다. 성능평가의 비교로 제시된 단순 할당과 정적 할당은 환경의 변화가 거의 없어 수행능력에 차이가 없는 경우에도 단순 할당에 비해 52.61%의 성능향상을 보였으며, 특히 성능변화가 극심한 경우에는 단순 할당과 정적 할당에 비해 각각 66.77%, 36.94% 정도 성능향상 되었다. 심지어 호스트의 고장이 발생했을 경우에는 정적 할당의 경우는 수행시간은 무한대가 되어 결과를 얻을 수 없게 되었으며 고장이 없는 경우에 비해 22.04%가 높은 처리시간을 가지지만 결과에 대한 응답은 얻을 수 있었다.

향후 과제로 고려되어야 사항은 첫째로 동적인 작업 처리자 관리에 관한 연구이다. 본 논문에서는 정적으로 고정된 참여 호스트를 대상으로 연산을 수행하였으며 현재 참여한 작업 처리자의 추가에 대해 어떠한 영향을 받지 않았으며 작업 처리자 삭제는 수행중 고장을 판단함으로써 작업 처리자를 관리하는데 그쳤으므로 이러한 제한점을 극복하기 위한 동적인 작업 처리자 관리 모델 연구가 요구된다. 둘째로 애플리케이션의 다양화이다. WebImg 시스템 구성 요소들과 애플리케이션으로 활용한 프랙탈 이미지 처리와의 연결을 구현된 인터페이스 클래스에 의해 이루어지며 이러한 응용으로만 성능평가를 마쳤다. 다양한 애플리케이션을 사용하여 제안된 작업 할당 기법을 동일한 환경에서 비교 성능평가가 필요하다.

참 고 문 헌

[1] A. Baratloo, M. Karaul, H. Karl and Z. M. Kedem, "An Infrastructure of Network Computing with Java Applets", *the ACM Workshop on Java High-Performance Network Computing*, 1998.

[2] K. M. Chandy, B. Dimitrov, H. Le, J. Mandleson, M. Richardson, A. Rifkin, P. A. G. Sivilotti, W. Tanaka, and L. Weisman, "A World-Wide Distributed System Using Java and the Internet", In *Proc. of the 5th IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, 1996.

[3] G. Fox and W. Furmanski, "Computing on the Web-New Approaches to Parallel Processing Petaop and Exaop Performance in the Year 2007", 1997

[4] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer, "ATLAS : An Infrastructure for Global Computing", In *Proc. of the 7th ACM SIGOPS European Workshop: System Support for Worldwide Application*, 1996.

[5] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wychoff, "Charlotte:Meta-computing on the Web", *The 9th International Conference on Parallel and Distributed Computing Systems*, Dijon, France, 1996.

[6] T. Brecht, H.sandhu, M. Shan, and J. Talbot, "ParaWeb: Towards World-Wide Supercomputing", In *Proc. of the 7th ACM SIGOPS European workshop : System Support for Worldwide Application, Connemara*, Ireland, 1996.

[7] A. D. Alexandra, M. Ibel, and K. E. Schauser, and C. J. Scheiman, "SuperWeb : Towards a Global Web-based Parallel Computing Infrastructure", *11th International Parallel processing Symposium*, 1997.

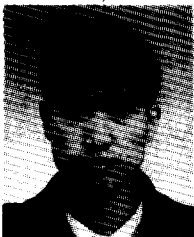
[8] N. Camiel, S. London, N. Nisan, O. Regev. "The POPCORN Project:Distributed computing over the Internet in Java", In *Proc. 6th International World Wide Web Conference*, 1997.

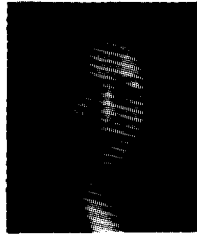
[9] B. O. Christiansen, P. Cappello, M. F. Ionescu,

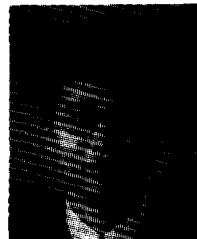
M. O. Neary, K. E. Schauser, D. Wu, "Javelin: Internet-Based Parallel Computing Using Java", *ACM Workshop on Java for Science and Engineering Computation*, 1997.

[10] Hernani Pedroso, Luis M. Silva, Victor Batista, Paulo Martins, Guilherme Soares and Telmo Menezes, "Web-based Metacomputing with JET", In *Proc. of Concurrency:Practice and Experience*, 1997.

[11] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1997.

[12] J. Dongarra, Performance of Various Computers Using Standard Linear Equation Software, http://www.netlib.org/benchmark/performance.ps, 1998.

정 권 호

1997년 원광대학교 컴퓨터공학과(공학사)
1999년~현재 원광대학교 컴퓨터공학과 석사과정
관심분야 : 병렬분산시스템, 이동컴퓨팅, 멀티미디어 CAI
E-mail: khjung@gaebyok.wonkwang. ac.kr

송 은 하

1997년 원광대학교 통계학과(이학사)
1998년~현재 원광대학교 컴퓨터공학과 석사과정
관심분야 : 병렬분산시스템, 병렬처리, 멀티미디어 CAI
E-mail: ehsong@gaebyok.wonkwang.ac.kr

정 영 식

1987년 고려대학교 수학과(이학사)
1989년 고려대학교 전산과학과(이학석사)
1993년 고려대학교 전산과학과(이학박사)
1998년 미시간주립대학교 교환교수
1993년~현재 원광대학교 컴퓨터 및 정보통신 공학부 부교수
관심분야 : 병렬분산시스템, 컴퓨터 시뮬레이션, 이동컴퓨팅, 멀티미디어 CAI
E-mail: ysjeong@wonkwang.ac.kr